

How to attack (and secure) an Android app: An introduction





Caner Kaya

caner.kaya@promon.no

PROMON

Who am I

- **Expertise:**
 - Android Security
 - Reverse Engineering
 - Vulnerability Analysis
 - Malware Analysis
- **Hobbies:**
 - Mountain Biking 
 - Snowboarding 
- **Mission:** Securing Apps & Shredding Slopes!
- **Motto:** "I code with one hand, hack with the other, and balance on two wheels in between!"

Disclaimer: Do not do this on your girlfriend/boyfriend's device

Introduction

- What is this workshop about?
 - Showing the view of an attacker.
 - Mostly practical demonstrations.
 - Discussion of countermeasures.
- Material: <https://github.com/pplithium/droidcon-lisbon>



Reverse engineering

- Understanding how an app works.
- Reveal secrets in it.
- First step of an attacker.
- Two complementary approaches: Static and dynamic
- On Android
 - Java code (Java, Kotlin)
 - Native code (C, C++, Dart, ...)

Reverse engineering Java code

Code in classes.dex file(s).

Dalvik bytecode executed in VM.

Requires disassembler¹ or decompiler².

¹ <https://github.com/iBotPeaches/Apktool>

² <https://github.com/skylot/jadx>



Demo



PROMON

Protecting against reverse engineering

- Impossible to prevent.
- Obfuscation can make it harder.
- Some things you can do
 - Rename/remove identifiers.
 - Encrypt strings.
 - Use reflection.
 - Use native code.
- Ideally done with a tool^{1,2,3,4,5}.

¹ <https://r8.googlesource.com/r8>

² <https://www.guardsquare.com/proguard>

³ <https://github.com/obfuscator-llvm/obfuscator>

⁴ <https://obfuscator.re/omvll>

⁵ <https://obfuscator.re/dprotect>

Repackaging

- Modifying app on disk.
- Change code to change behavior.
- Change resources to change look.

Patching Java code

- Modify classes.dex file(s).
- Direct binary patching can be tricky.
- Tools like apktool make this easy
 - Disassemble to smali.
 - Modify smali.
 - Re-assemble to apk.



Demo



PROMON

A real scenario – Introduction

- A company for door systems vulnerable in both app and the door computer
- We had reverse-engineered our way into the building.
 - Reverse engineered the application.
 - Repackaged the application and tracked the communication.
 - Instrumented the communication on our fake application, and added our NFC tags for free entrance.

A real scenario – Part 1

- We have firstly analyzed the apk to find the NFC channels used (*mAID*). It was quite openly shown in the code.

```
.line 166
const/4 v1, 0x1

new-array v1, v1, [Ljava/lang/String;

const/4 v2, 0x0

const-string v3, "4E45574754"

aput-object v3, v1, v2

invoke-static {v1}, Ljava/util/Arrays;->asList([Ljava/lang/Object;)Ljava/util/List;

move-result-object v1

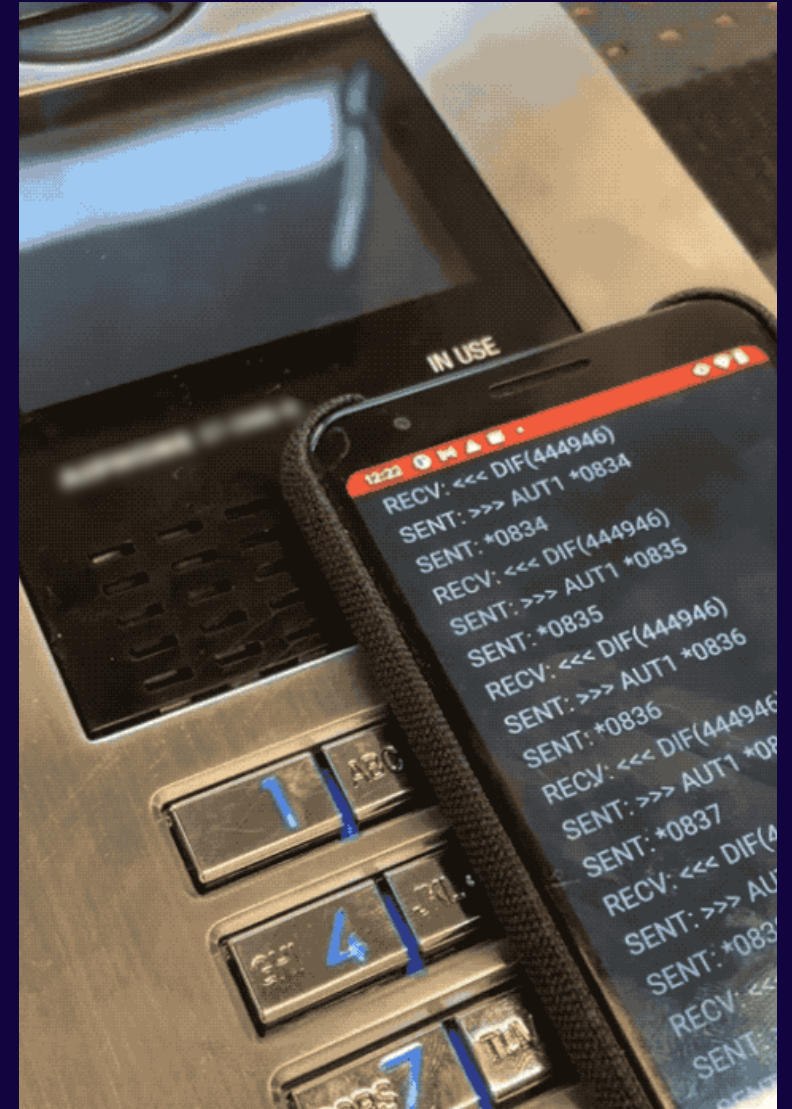
.line 167
iget-object v2, p0, Ljp/co/aiphone/ngt_android_setting_tool/NFCCommunicationActivity;-->A:Landroid/nfc/cardemulation/CardEmulation;

const-string v3, "other"

invoke-virtual {v2, v0, v3, v1}, Landroid/nfc/cardemulation/CardEmulation;-->registerAidsForService(Landroid/content/ComponentName;Ljava/lang/String;Ljava/util/List;)Z
```

A real scenario – Part 2

- With the NFC Channel, we attempted to communicate with the NFC based door system.
Door system:
 - there must be a management system with an admin user for managing
 - there must be the 'normal users' for opening the doors
- We have first tried brute forcing the admin password:



A real scenario – Part 3

- After finding the admin password with a brute force attack, we attempted to send our 'fake' NFC tags to attempt to inject them into the door system database.
- However, things weren't as easy as it was until now.
- To solve the problem, we have injected a logger method between the original app after repackaging it, and we recompiled it to watch the communication.

A real scenario – Part 3

```
.method public logMe(Ljava/lang/String;)V
    .registers 4
    .param p1, "str"    # Ljava/lang/String;
    .annotation system Ldalvik/annotation/MethodParameters;
        accessFlags = {
            0x0
        }
        names = {
            "str"
        }
    .end annotation

    .line 384
    new-instance v0, Ljava/lang/StringBuilder;

    invoke-direct {v0}, Ljava/lang/StringBuilder;-><init>()V

    const-string v1, "comm: "

    invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    move-result-object v0

    invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;

    move-result-object v0

    const-string v1, "NFC_COMMAND"

    invoke-static {v1, v0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    .line 385
    return-void
.end method
```

```
.method public processCommandApu([BLandroid/os/Bundle;)[B
    .registers 11
    .param p1, "commandApu"    # [B
    .param p2, "extras"    # Landroid/os/Bundle;
    .annotation system Ldalvik/annotation/MethodParameters;
        accessFlags = {
            0x0,
            0x0
        }
        names = {
            "commandApu",
            "extras"
        }
    .end annotation

    .line 149
    new-instance v0, Ljava/lang/String;

    sget-object v1, Ljava/nio/charset/StandardCharsets;->US_ASCII:Ljava/nio/charset/Charset;

    invoke-direct {v0, p1, v1}, Ljava/lang/String;-><init>([BLjava/nio/charset/Charset;)V

    .line 151
    .local v0, "s":Ljava/lang/String;
    invoke-static {p1, Landroid/os/Bundle; /services/6THostApuInjectorService;->bytesToHex([B)Ljava/lang/String;

    move-result-object v1

    invoke-virtual {p0, v1}, Landroid/os/Bundle; /services/6THostApuInjectorService;->logMe(Ljava/lang/String;)V
    .line 152
    .local v1, "hex":Ljava/lang/String;
    .end method
```

A real scenario – Part 4

- We have developed an application that does the exact communication for adding and removing the users.
- We extended this attack as an application to:
 - Find the admin code
 - Add the hacker user (NFC Tag)
 - Enter to the building
 - Remove the user
 - Clear all the traces

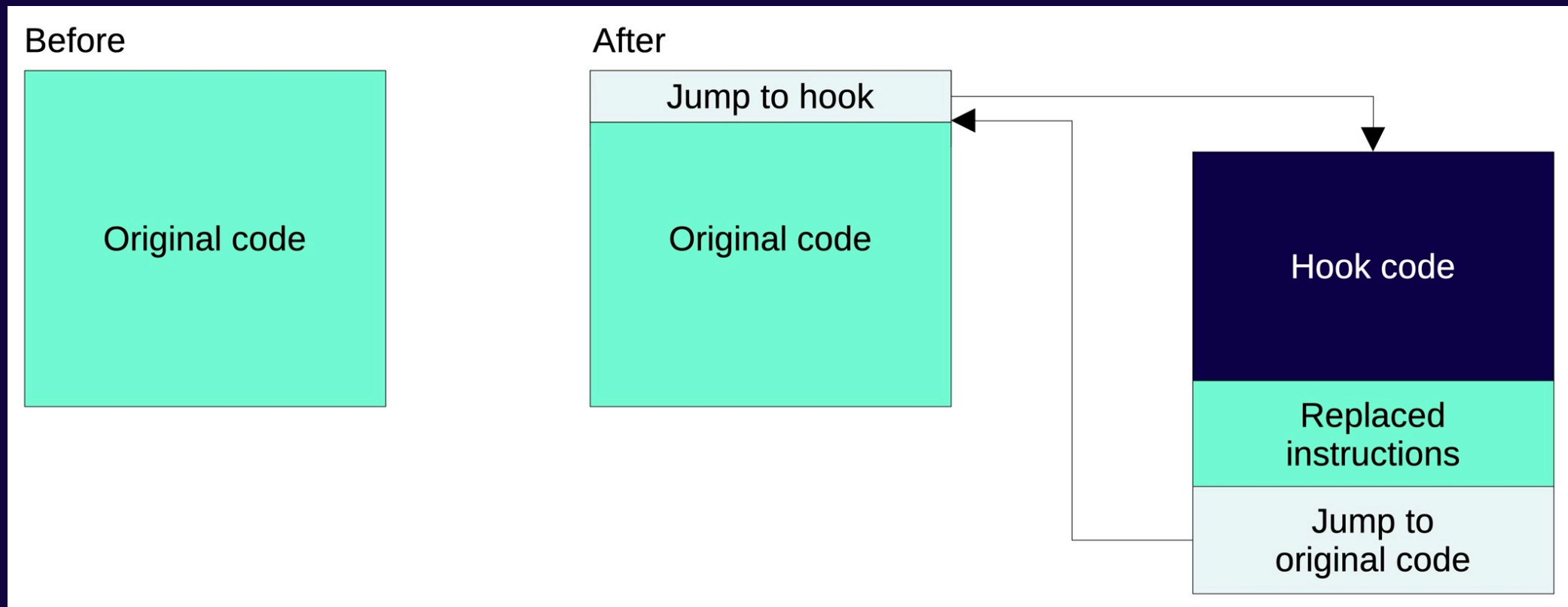
Protecting against Repackaging

- Implement anti-tampering mechanisms
 - Check APK signature and signer.
 - Implement checksumming mechanism.
- Can also be patched.
- Obfuscation can make this more difficult.
- Multiple independent mechanisms can make this more difficult.

Hooking

- Modify the app while it runs.
- Change code to change behavior.
- Useful for dynamic reverse engineering.

How hooking works



Hooking Java code

Code is executed in VM.

Could be compiled ahead of time or just in time.

Requires modifying the VM.

Popular hooking frameworks

LSPosed

Frida

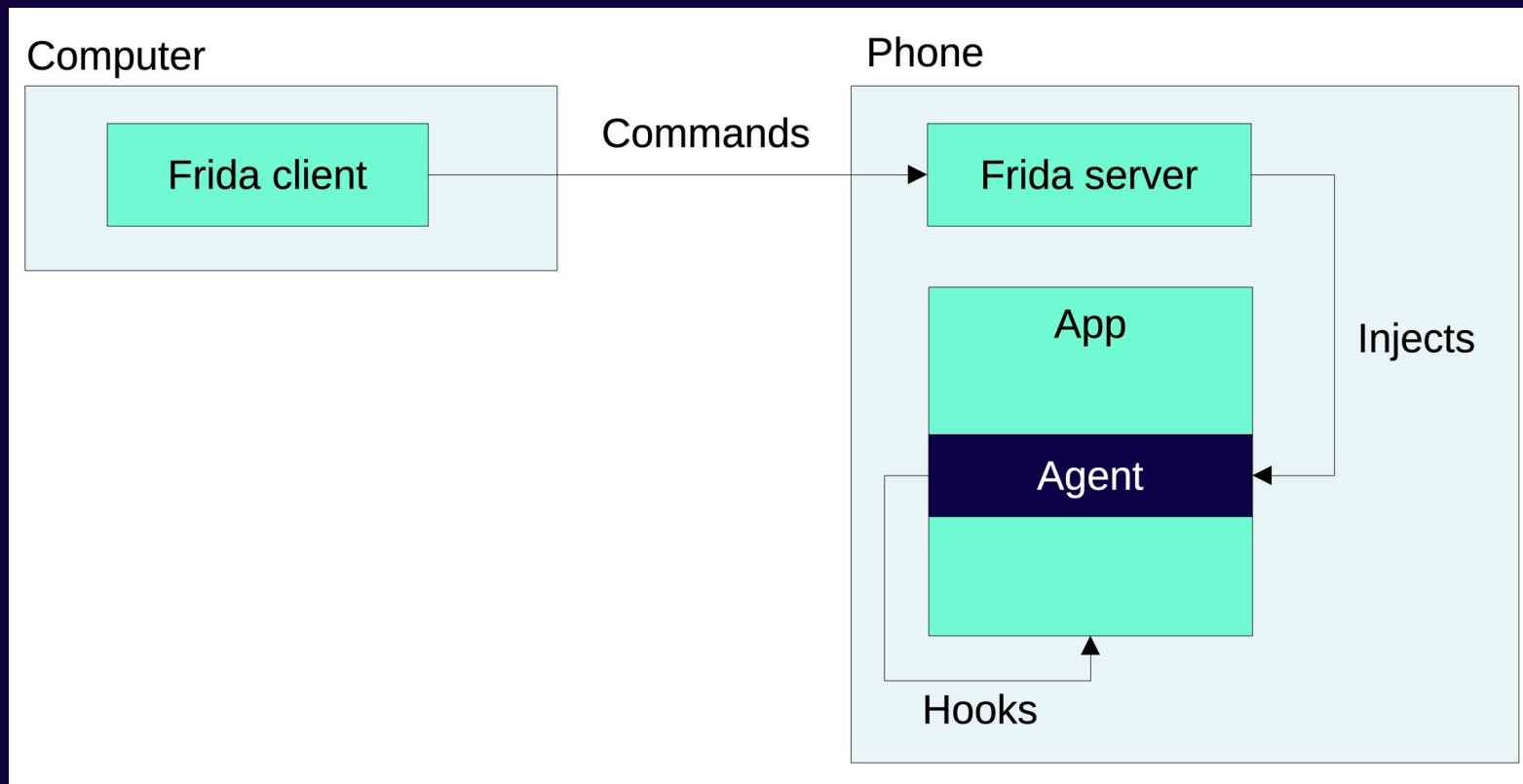


Hooking native code

- Overwrite code in memory.
- Not completely trivial.
- Frida is a popular framework to use.

The logo for Frida, featuring the word "FRIDA" in a bold, red, sans-serif font. The letters are slightly stylized, with the 'F' and 'A' having a more geometric, blocky appearance. The logo is centered within a white rectangular box.

How Frida works in our use case



Demo



PROMON

Protecting against Hooking

- Detect hooks
 - Check for code modifications in memory.
- Detect hooking framework
 - Check for suspicious files, libraries and communication channels.
- Can also be hooked.
- Obfuscation and multiple independent mechanisms make it harder.

Strandhogg

- Strandhogg Attack has been discovered in 2019
- Niche
- Identified *non-disclosed* amount of malicious apps in the wild
- It uses taskAffinity attribute, and gets injected into another app's Task*(1)

<https://developer.android.com/topic/security/risks/strandhogg>

Strandhogg

- What means Task* in this context?

Partial Demo

Demo



PROMON

Summary

- Is this a problem for you?
- Possible to implement countermeasures yourself.
- Better than doing nothing but probably not too effective.
- It might be worth considering getting help.



Thank you!



Caner Kaya
Software Engineer
caner.kaya@promon.no